UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/550,728	09/26/2005	Yuji Nadamoto	2005_1474A	6062

52349 7590 03/25/2008

WENDEROTH, LIND & PONACK L.L.P.
2033 K. STREET, NW
SUITE 800
WASHINGTON, DC 20006

EXAMINER	
HUR, ECE	

ART UNIT	PAPER NUMBER
2179	

MAIL DATE	DELIVERY MODE
03/25/2008	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Office Action Summary	Application No.	Applicant(s)	
	10/550,728	NADAMOTO, YUJI	
	Examiner	Art Unit	
	ECE HUR	2179	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE *3* MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *14 December 2007*.

2a)☒ This action is **FINAL**. 2b)☐ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *17-38* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *17-38* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☒ The drawing(s) filed on *26 September 2005* is/are: a)☒ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☒ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☒ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☐ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☐ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date _____.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____.
5) ☐ Notice of Informal Patent Application
6) ☐ Other: _____.

DETAILED ACTION

This action is responsive to Remarks/Arguments filed on December 14, 2007. Claims 17-38 are amended and pending in this application. This application is a new PCT National Stage application of PCT/JP04/04414 that was filed on March 29, 2004. Applicant is claiming foreign priority for the application 2003-106393 filed on April 10, 2003 in Japan.

Status of Claims

Claims 1-17 are cancelled in the case.
Claims 17-38 are pending in the case. Claims 17, 37 and 38 are independent claims.
Claim 37 is rejected under 35 U.S.C. 101.
Claims 17-38 are rejected under 35 U.S.C. 103(a).

Response to Arguments

Applicant's arguments filed December 14, 2007 have been fully considered but they are not persuasive. See rejection details. Applicant argued:

1) Regarding Claim 37 rejection under 35 U.S.C. § 101 as being directed to non-statutory subject matter remains because Claim 37 still claims the window management program not the recording medium.

2) Applicant argued that Ashe does not teach the claimed aspect of updating the stack order of a window within a window group without altering a stacking order of the groups of windows within the stack of windows. However, it would be

obvious to one of ordinary skill in the art the time of the invention to update the window order within the stack without modifying the stacking order of the other groups, because this would provide modularity and a system composed of modules that each serve a specific purpose in this case display a window within a group and communicate with each other to produce the system's overall behavior.

3) Amendment to specification has been placed in the application file.

4) Applicant further has remarks about the amended Claims.

Information Disclosure Statement Acknowledgement

The information disclosure statements filed on September 26, 2005 is in compliance with the provisions of 37 CFR 1.97, 1.98 and MPEP § 609. It has been placed in the application file, the information referred to therein has been considered as to the merits.

Priority Acknowledgement

Acknowledgment is made of applicant's claim for foreign priority under 35 U.S.C. 119(a)-(d). Receipt is acknowledged of certified copy of Application No.2003-106393, Japan, filed on April 10, 2003 submitted under 35 U.S.C. 119(a)-(d), which papers have been placed of record in the file.

Specification Objection

The lengthy specification has not been checked to the extent necessary to determine the presence of all possible minor errors. Applicant's cooperation is requested in correcting any errors of which applicant may become aware in the specification.

Claim Rejections - 35 USC § 101

35 U.S.C. 101 reads as follows:

> Whoever invents or discovers any new and useful process, machine, manufacture, or composition of matter, or any new and useful improvement thereof, may obtain a patent therefor, subject to the conditions and requirements of this title.

Claim 37 is rejected under 35 U.S.C. 101 because the claimed invention is directed to non-statutory subject matter, specifically directed towards Software per se.

Regarding Claim 37, Claim 37 is rejected under 35 U.S.C. 101 because the claimed invention is directed to non-statutory subject matter and claiming "Software" per se. Software is functional descriptive material that can be considered statutory only if it is both functional and clearly embodied on a computer readable medium and designed to support specific data manipulation function. When functional descriptive material is recorded on a computer-readable medium it will become structurally and functionally interrelated the medium and will be statutory in most cases since the use of technology permits the function of the descriptive material to be realized. See In re Lowry, 32 F.3D 1579, 32 USPQ2d 1031, 1035 (Fed. Cir 1994)

and Warmerdam, 33 F.3d at 1360-61, 31 USPQd at 1759. A Software structure is functional if the specific arrangement of data enables a computer to accomplish useful result arising from the arrangement of the data in the software. However, only computer readable medium executed instruction by a processor could be statutory, it is not clearly defined as being embodied in a computer readable medium as executed instruction and is therefore not statutory. See Warmerdam, 33 F.3d at 1360, 31 USPQ2d at 1759.

Claim Rejections - 35 USC § 103

The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

Claims 17-38 are rejected under 35 U.S.C. 103(a) as being obvious over Ashe, US Patent 5,995,103.

Regarding Claim 17, Ashe discloses the claimed aspect of a window stack control method for managing a stack of a plurality of windows which are displayed on a display unit based on one or more application programs, wherein a window grouping

mechanism for creating, organizing and manipulating windows and, more specifically, window groups for display to a user of a computer system is disclosed, by using a series of linked data structures configured to organize information pertaining to all windows defined by an application program executing on the computer system. (See Abstract).

Ashe discloses the claimed aspect of receiving a request for newly creating a window from the application program in FIG. 2, wherein an application program 202 executes on the computer system 200. The application program 202 and the operating system 204 interact (via arrow 206), via task commands of an application programming interface (API) layer 207, to control the operations of the computer system 200. (Ashe, Page 6, Paragraphs 5,10). Furthermore the operating system contains system facilities, including a window manager 205 which, inter alia, directly implements those task commands. (Ashe, Page 6, Paragraph 15 lines 1-3).

Ashe discloses the claimed aspect of receiving a request for displaying the window from the application program, wherein the window manager 205 is a set of software routines within the operating system 204 that is responsible for managing windows 242 displayed on a display screen 235 and viewed by the user during operation of the application program 202. The window manager 205 typically acts in direct response to task commands sent from the application program 202 to the operating system 204 via the API layer 207 as shown by arrow 203. (Ashe, Page 6, Paragraph 30). More specifically, the window manager 205 may further use a graphics system 209 within the operating system to draw the display screen 235. The graphics system 206 stores the

information to be displayed directly (via arrow 208) into a screen buffer 210. Under the control of various hardware and software in the computer system 200, the contents of the screen buffer 210 are read out of the buffer 210 and provided, as indicated schematically by arrow 212, to a display adapter 214. The display adapter 214 contains hardware and software (sometimes in the form of firmware) which converts the information in the screen buffer 210 to a form which can be used to drive the display screen 235 of monitor 232, which is connected to the display adapter 214 by an adapter cable 218. (Ashe, Page 6, Paragraphs 35-45).

Ashe discloses the claimed aspect of designating a group of the window from the application program in FIG. 7 and FIG. 8, wherein window grouping is illustrated.

Ashe discloses the claimed aspect of collectively arranging the window as a group so as to determine a stacking order of the window in the group in response to the display request, a the stacking order of the window is determined so as to be consecutively followed by a stacking order of a group to which the window belongs, with maintaining stacking orders of respective window groups of windows which have been collectively arranged as groups in FIG. 7 and FIG. 8, wherein the Master Application Window List 700 comprises a singly linked list of data elements 710 wherein each data element 710 corresponds to a window 306 (FIG. 3) created by the application program 302. Each data element 710, moreover, is preferably arranged as a table array 720 comprising a series of entries 730 containing fields for storing information related to that window 306. For example, the first entry 731 identifies the class of window being represented, e.g., document, floating or modal. The second entry 732 contains the

information regarding the window's appearance as defined by the application program 302. The second entry 732, rather than containing the actual data supporting the window 306, may instead contain a pointer to another location in memory 114 (FIG. 1) where that information is stored. (Ashe, Page 9, Paragraphs 25-40). The third entry 733 in the table array 720 contains a value indicating the number of window groups to which this window 306 belongs. The significance of this entry 733 is explained in more detail below. The fourth entry 734 signifies whether any window groups are attached to the window 306 and, if so, the identification and memory location of that window group, again, as explained below. The fifth entry 735 in each table array 720 is a forward link to the next element 710 in the list 700 as shown by arrow 738 and the last entry 736 contains a mark that may be either set or unset during window operations. (Ashe, Page 10, Paragraphs 45-50).

Furthermore, Ashe discloses that the application program 302 (FIG. 3) creates a window 306, it can specify where in the list 700 that window element 710 is placed and thus in what order the window will be displayed relative to the other defined windows. (Ashe, Pages 9-10, Paragraph 5, lines 1-4).

Ashe does not teach the claimed aspect of updating the stack order of a window within a window group without altering a stacking order of the groups of windows within the stack of windows. It would be obvious to one of ordinary skill in the art the time of the invention to update the window order within the stack without modifying the stacking order of the other groups, because this would provide modularity and a system composed of modules that each serve a specific purpose in this case display a window

within a group and communicate with each other to produce the system's overall behavior.

Regarding Claim 18, most of the limitations have been met in the rejection of claim 17. See details for Claim 17 rejection. Ashe discloses the claimed aspect of creating a representative window for each group as collectively arranging step includes handling, when collectively arranging a window as a group so as to determine a stacking order of the window in the group, the window as a child window of the representative window in FIG. 5, wherein a simplified class hierarchy diagram for the window objects is illustrated. The class HIWindow 502, used to construct the (parent) window object, is a subclass of an abstract base class called HIObject 500. Notably, this class 500 is a base class for all window classes and provides all message-handling functions that a window class needs. Each of the classes used to construct the (child) window objects are subclasses of HiWindow and thus inherit any functional operators that are available from that class. For example, the subclass HIModal 504 is derived from HIWindow 502 and encapsulates the modal window. Similarly, the HIFloating class 506 and the HIDocument class 508 are used to construct floating windows and document windows, respectively. Hereinafter, we will refer to window objects 306 (FIG. 3) simply as windows. (Ashe, Page 8, Paragraphs 45-55).

Regarding Claim 19, most of the limitations have been met in the rejection of claim 18. See details for Claim 18 rejection. Ashe achieves the claimed aspect of receiving a request for shifting a top of a group in the window stack or a request for shifting a bottom of a group in a window stack, from the application program and changing the stack in the group in response to the request for shifting the top or the request for shifting the bottom in SelectWindowWithGroup () function and Select () function, wherein creating window groups and adding window groups to groups, the application program 302 can also cause the display screen 235 to be re-drawn with a selected window and window group at the front of the screen 235, by invoking the SelectWindowWithGroup method. This method preferably uses the references for the selected window and window group as its arguments. (Ashe, Page 14, Paragraph 5).

Furthermore in response to invocation of this method, the window element 710 corresponding to the selected window is accessed and the mark in its sixth entry 736 is set. Next, the group element 802 corresponding to the selected group is accessed first to set the mark contained in the seventh entry, signifying that this group has been addressed. After setting the mark, the address pointers listed in fourth entry 824 are scanned to identify each window element 710 corresponding to each window added to the selected group. Each element 710 is then sequentially addressed and the mark contained in its sixth entry 736 is set. (Ashe, Page 14, Paragraph 10-15)

Regarding Claim 20, most of the limitations have been met in the rejection of claim 19. See details for Claim 19 rejection. Ashe discloses the claimed aspect of

changing of the includes changing the stack so as to collect, when receiving, from the application program, the request for shifting a first window to the top of the group in the window stack or the request for shifting the first window to the bottom of the group in the window stack, a first window group of windows which belong to a same group as the first window and which are not collectively arranged as a group and a second window group of windows which belong to the same group as the first window and which have been collectively arranged as a group such that a stacking order of the first window group is consecutively followed by a stacking order of the second window group in FIG. 8, wherein a diagram depicting an illustrative arrangement of data structures 802 in a portion of memory 114 (FIG. 1) that are created and manipulated in response to a set of functions. It should be noted that these methods are preferably associated with the window objects 306 constructed by the application program 302. However, in an alternative embodiment, the application programs may issue these functions as task commands (system calls) embedded within the API layer of the computer system. (Ashe, Page 11, Paragraph 30).

Furthermore, Ashe discloses the claimed aspect of shifting the first window to a top of a group in a window stack and the request for shifting the first window to a bottom of a group in a window stack in GetWindowsInWindowGroup function, wherein to review the windows in a particular window group, the GetWindowsInWindowGroup method is invoked using the reference for the selected window group as the argument. In response, the element 802 corresponding to the selected group is accessed in order to examine the data field of the fourth entry 824 which provides a list of the windows that

have been added to the selected group. More specifically, the entry 824 provides a list of address pointers to the window elements 710 in the Master Application Window List 700. These address pointers are used to access each window element 710 in list 700 and set the mark contained in the sixth entry 736. (Ashe, Page 16-17, Paragraph 65).

Ashe does not teach the claimed aspect of a stacking order of the first window group is not altered and is consecutively followed by a stacking order of the second window group, however this would be obvious to one of ordinary skill in the art at the time of the invention because this would provide modularity and a system composed of modules that each serve a specific purpose in this case display a window within a group and communicate with each other to produce the system's overall behavior. Furthermore, Ashe discloses in FIG. 7 illustrates different groups of windows such as Modal Windows, 752, Floating Windows 754, Document Windows 756.

Regarding Claim 21, most of the limitations have been met in the rejection of claim 18. See details for Claim 18 rejection. Ashe discloses the claimed aspect of receiving a request for shifting a top of the groups of windows in the window stack or a request for shifting a bottom of the groups in the window stack, from the application program and changing a stacking of the groups in response to the request for shifting the top or the request for shifting the bottom in SelectWindowWithGroup () function and Select () function, wherein creating window groups and adding window groups to groups, the application program 302 can also cause the display screen 235 to be re-drawn with a selected window and window group at the front of the screen 235, by

invoking the SelectWindowWithGroup method. This method preferably uses the references for the selected window and window group as its arguments. (Ashe, Page 14, Paragraph 5).

Furthermore in response to invocation of this method, the window element 710 corresponding to the selected window is accessed and the mark in its sixth entry 736 is set. Next, the group element 802 corresponding to the selected group is accessed first to set the mark contained in the seventh entry, signifying that this group has been addressed. After setting the mark, the address pointers listed in fourth entry 824 are scanned to identify each window element 710 corresponding to each window added to the selected group. Each element 710 is then sequentially addressed and the mark contained in its sixth entry 736 is set. (Ashe, Page 14, Paragraph 10-15). Furthermore, the Master Application Window List 700 is then scanned using the forward links 735 in each window element 710 to identify all "marked" windows (i.e., window elements 710 containing set marks in their sixth entry 736). The marked windows are then moved from their current locations in the Master Window Application List 700 to the front of their respective class sequences 752, 754, 756. During this "re-ordering" exercise, the marked windows are re-drawn to the front of the display screen 235. The Master Application Window List 700 is then scanned using the forward links in the fifth entries 735 of the window elements 710 and the marks in the sixth entries 736 are reset. Similarly, the list of window groups 800 is scanned using the forward links of the group elements 802 and the marks in the seventh entries 827 of the window elements 802 are

reset; as a result, the Master Application Window List 700 and the list of window groups 800 are ready for the next operation. (Ashe, Page 14, Paragraphs 45-55).

Regarding Claim 22, most of the limitations have been met in the rejection of claim 21. See details for Claim 21 rejection. Ashe discloses the claimed aspect of changing step of the stacking of the groups includes changing a the stack so as to collect, when receiving, from the application program, the request for shifting a first group to a the top of the groups of windows in the window stack and or the request for shifting a the first group to a the bottom of the groups of windows in the window stack, a first window group of windows which belong to the first group and which are not collectively arranged as a group and a second window group of windows which belong to the first group and which have been collectively arranged as a group such that the stacking order of the first window group is not altered and is consecutively followed by a stacking order of the second window group in GetWindowGroupsInWindowGroup function, wherein to view the window groups in a particular group, GetWindowGroupsInWindowGroup function is invoked using the group reference as an argument. Here, the second entry 822 of each data element 802 corresponding to the selected group is accessed to determine whether any groups are currently encompassed in the selected group. If the selected group encompasses other groups, the list of groups contained in entry 825 is copied into a portion of buffer memory accessible by the application program 302. The application program 302 can then review this list of group references to determine which groups are encompassed in the selected group. (Ashe, Page 17, Paragraphs 60-65).

Regarding Claim 23, most of the limitations have been met in the rejection of claim 21. See details for Claim 21 rejection. Ashe discloses the claimed aspect of an X window system and a window manager manage the stack of a plurality of windows, and a specific window is disposed immediately above a group at the top in DisposeWindowGroup () function, wherein the DisposeWindowGroup method is employed using the reference for the disposed group as the argument. In response to this method, information contained within the third entry 823 in the table array 804 of the disposed window group is reviewed to determine if the disposed window group is a member of any other window groups. If the value within this entry 823 is greater than zero, then the disposed group has been added to other groups and those groups must be accessed to complete the method. In particular, the pointer contained in the fifth entry 825 of each data element 802 corresponding to those window groups associated with the disposed group is deleted; the value of the second entry 822 is then decremented; and the portion of memory previously allocated for the disposed group is deallocated and returned to the memory management facility. (Ashe, Page 13, Paragraphs 55-65).

Regarding Claim 24, most of the limitations have been met in the rejection of claim 18. See details for Claim 18 rejection. Ashe achieves the claimed aspect of X window system and a window manager manage the stack of a the plurality of windows, and the window manager confirms whether or not its a recognized stack conforms to a

stack recognized by the window system when receiving a window destruction notification, and, conformity when the recognized stack of the window manager does not conform to the stack recognized by the window system, the window manager performs processing for conforming a the stack recognized by the window system to a the stack recognized by the window manager in RemoveFromGroup () function, wherein to remove a window from a window group, the RemoveFromGroup method is preferably invoked using the window reference and the window group reference as its arguments. By invoking the RemoveFromGroup method, the address pointer of the removed window is deleted from the fourth entry 824 in the table array 804 associated with the selected window group. The first entry 821 in the table array 804 is then decremented to reflect the current number of windows in the group. Finally, the element 710 corresponding to the removed window is accessed in order to decrement the third entry 733 signifying the current number of window groups to which the window belongs. (Ashe, Page 13, Paragraphs 5-15). Furthermore, Ashe discloses a RemoveWindowGroupFromWindowGroup (); function, wherein the RemoveWindowGroupFromWindowGroup method preferably functions to remove a window group from another window group using, as arguments, the references for the base group and the target group. Here, invocation of the method results in the address pointer for the target window being deleted from the fifth entry 825 in the table 804 corresponding to the base group. In addition, the value of the second entry 822 for the base group is decremented signifying that the base group contains one less window group. Finally, the data element 802 corresponding to the target window group is

accessed and the value of the third entry 823 is decremented. (Ashe, Page 13, Paragraphs 40-50).

Regarding Claim 25, most of the limitations have been met in the rejection of claim 18. See details for Claim 18 rejection. Ashe achieves the claimed aspect of an X window system and a window manager manage a the stack of a the plurality of windows, and the window manager sets a flag when requesting the window system to change the stack, and confirms whether or not its a recognized stack conforms to a stack recognized by the window system only when the flag is set at a reception of a window destruction notification, and, when the recognized stack of the window manager does not conform to the stack recognized by the window system, the window manager performs processing for conforming the stack recognized by the window system to the stack recognized by the window manager, to thereby to put the flag down in FIG. 7 and FIG. 8, wherein Mark 736 and Mark 827 are illustrated that may be either set or unset during window operations. Applicant should duly note that Mark has a flag function.

Regarding Claim 26, most of the limitations have been met in the rejection of claim 17. See details for Claim 17 rejection. Ashe achieves the claimed aspect of receiving a request for shifting a top of a group in a window stack and a request for shifting a bottom of a group in a window stack, from the application program and changing the stack in the group in response to the request for shifting the top or the request for shifting the bottom in SelectWindowWithGroup () function and Select ()

function, wherein creating window groups and adding window groups to groups, the application program 302 can also cause the display screen 235 to be re-drawn with a selected window and window group at the front of the screen 235, by invoking the SelectWindowWithGroup method. This method preferably uses the references for the selected window and window group as its arguments. (Ashe, Page 14, Paragraph 5).

Furthermore in response to invocation of this method, the window element 710 corresponding to the selected window is accessed and the mark in its sixth entry 736 is set. Next, the group element 802 corresponding to the selected group is accessed first to set the mark contained in the seventh entry, signifying that this group has been addressed. After setting the mark, the address pointers listed in fourth entry 824 are scanned to identify each window element 710 corresponding to each window added to the selected group. Each element 710 is then sequentially addressed and the mark contained in its sixth entry 736 is set. (Ashe, Page 14, Paragraph 10-15).

Regarding Claim 27, most of the limitations have been met in the rejection of claim 26. See details for Claim 26 rejection. Ashe discloses the claimed aspect of the changing of the stack includes changing the stack so as to collect, when receiving, from the application program, the request for shifting a first window to a the top of the group in a the window stack and or the request for shifting a the first window to a the bottom of the group in the window stack, a first window group of windows which belong to a same group as the first window and which are not collectively arranged as a group and a second window group of windows which belong to the same group as the first window and which have been collectively arranged as a group such that a stacking

order of the first window group is consecutively followed by a stacking order of the second window group in GetWindowGroupsInWindowGroup function, wherein to view the window groups in a particular group, GetWindowGroupsInWindowGroup function is invoked using the group reference as an argument. Here, the second entry 822 of each data element 802 corresponding to the selected group is accessed to determine whether any groups are currently encompassed in the selected group. If the selected group encompasses other groups, the list of groups contained in entry 825 is copied into a portion of buffer memory accessible by the application program 302. The application program 302 can then review this list of group references to determine which groups are encompassed in the selected group. (Ashe, Page 17, Paragraphs 60-65).

Ashe does not teach the claimed aspect of first window not being altered and is consecutively followed by a stacking order of the second window group. It would be obvious to one of ordinary skill in the art the time of the invention to update the window order within the stack without modifying the stacking order of the other groups, because this would provide modularity and a system composed of modules that each serve a specific purpose in this case display a window within a group and communicate with each other to produce the system's overall behavior.

Regarding Claim 28, most of the limitations have been met in the rejection of Claim 17. See details for Claim 17 rejection. Ashe discloses the claimed aspect of

further comprising: receiving a request for shifting a top of the groups of windows in a the window stack or a request for shifting a bottom of the groups of windows in a the window stack, from the application program; and changing a stacking of the groups in response to the request for shifting the top or the request for shifting the bottom in SelectWindowWithGroup () function and Select () function, wherein creating window groups and adding window groups to groups, the application program 302 can also cause the display screen 235 to be re-drawn with a selected window and window group at the front of the screen 235, by invoking the SelectWindowWithGroup method. This method preferably uses the references for the selected window and window group as its arguments. (Ashe, Page 14, Paragraph 5).

Furthermore in response to invocation of this method, the window element 710 corresponding to the selected window is accessed and the mark in its sixth entry 736 is set. Next, the group element 802 corresponding to the selected group is accessed first to set the mark contained in the seventh entry, signifying that this group has been addressed. After setting the mark, the address pointers listed in fourth entry 824 are scanned to identify each window element 710 corresponding to each window added to the selected group. Each element 710 is then sequentially addressed and the mark contained in its sixth entry 736 is set. (Ashe, Page 14, Paragraph 10-15). Furthermore, the Master Application Window List 700 is then scanned using the forward links 735 in each window element 710 to identify all "marked" windows (i.e., window elements 710 containing set marks in their sixth entry 736). The marked windows are then moved from their current locations in the Master Window Application List 700 to the front of

their respective class sequences 752, 754, 756. During this "re-ordering" exercise, the marked windows are re-drawn to the front of the display screen 235. The Master Application Window List 700 is then scanned using the forward links in the fifth entries 735 of the window elements 710 and the marks in the sixth entries 736 are reset. Similarly, the list of window groups 800 is scanned using the forward links of the group elements 802 and the marks in the seventh entries 827 of the window elements 802 are reset; as a result, the Master Application Window List 700 and the list of window groups 800 are ready for the next operation. (Ashe, Page 14, Paragraphs 45-55).

Regarding Claim 29, most of the limitations have been met in the rejection of Claim 28. See details for Claim 28 rejection. Ashe discloses the claimed aspect of the stacking of the groups includes changing the stack so as to collect, when receiving, from the application program, the request for shifting a first group to the top of the groups of windows in the window stack and or the request for shifting a the first group to a the bottom of the groups of windows in the window stack, a first window group of windows which belong to the first group and which are not collectively arranged as a group and a second window group of windows which belong to the first group and which have been collectively arranged as a group with such that the stacking order of the first window group and is consecutively followed by a stacking order of the second window group in GetWindowGroupsInWindowGroup function, wherein to view the window groups in a particular group, GetWindowGroupsInWindowGroup function is invoked using the group reference as an argument. Here, the second entry 822 of each data

element 802 corresponding to the selected group is accessed to determine whether any groups are currently encompassed in the selected group. If the selected group encompasses other groups, the list of groups contained in entry 825 is copied into a portion of buffer memory accessible by the application program 302. The application program 302 can then review this list of group references to determine which groups are encompassed in the selected group. (Ashe, Page 17, Paragraphs 60-65).

Ashe does not teach the claimed aspect of updating the stack order of a window within a window group without altering a stacking order of the groups of windows within the stack of windows. It would be obvious to one of ordinary skill in the art the time of the invention to update the window order within the stack without modifying the stacking order of the other groups, because this would provide modularity and a system composed of modules that each serve a specific purpose in this case display a window within a group and communicate with each other to produce the system's overall behavior.

Regarding Claim 30, most of the limitations have been met in the rejection of Claim 28. See details for Claim 28 rejection. Ashe discloses the claimed aspect of an X window system and a window manager manage the stack of a plurality of windows, and a specific window is disposed immediately above a group at the top of the stack of windows in DisposeWindowGroup () function, wherein the DisposeWindowGroup method is employed using the reference for the disposed group as the argument. In response to this method, information contained within the third entry 823 in the table

array 804 of the disposed window group is reviewed to determine if the disposed window group is a member of any other window groups. If the value within this entry 823 is greater than zero, then the disposed group has been added to other groups and those groups must be accessed to complete the method. In particular, the pointer contained in the fifth entry 825 of each data element 802 corresponding to those window groups associated with the disposed group is deleted; the value of the second entry 822 is then decremented; and the portion of memory previously allocated for the disposed group is deallocated and returned to the memory management facility. (Ashe, Page 13, Paragraphs 55-65).

Regarding Claim 31, most of the limitations have been met in the rejection of Claim 17. See details for Claim 17 rejection. Ashe discloses the claimed aspect of a window which is designated as a priority window by the application program is not caused to belong to any group and the priority window is caused to be always disposed higher in stack than all the windows which are displayed on a display unit and which belong to any of the groups, wherein the window manager typically stores these windows in layers defined by and associated with these application programs. A window layer is simply a set of all the windows associated with a single application program. The window manager maintains these window layers in a block of memory. (Ashe, Page 1, Paragraph 60). In particular, a number is assigned to each layer of windows describing its priority class relative to the other window layers. For example, a window layer of priority class "2", e.g. a screen saver, will always appear in front of a window

layer of priority class "3", e.g. an application program. The window manager also controls the ordering in which document, floating and modal windows may appear within a specific application layer. (Ashe, Page 2, Paragraph 5-15, lines 1-12). Also, Trueblood, in US Patent 5,675755, achieves the claimed aspect of a priority window, wherein important information displayed in a window prevented becoming hidden to the user. For example, PC Tools for Windows provides a window-based graphical desktop that allows a user to specify that a window should remain "always on top". The "always on top" window is assigned a position in the stack order that is higher than the highest position assigned to ordinary windows. Consequently, if any window is manipulated in such a way as to overlap with the "always on top" window, the overlapping portion of the "always on top" window will cover the other window in the common screen region. (Trueblood, Page 1, Paragraph 5).

Regarding Claim 32, most of the limitations have been met in the rejection of Claim 17. See details for Claim 17 rejection. Ashe discloses the claimed aspect of a window disposed lower in stack of windows than a window belonging to a group at a top of the stack of windows is always in a state of non-display in FIG.6 A and B, wherein Sales report is not in full display.

Regarding Claim 33, most of the limitations have been met in the rejection of Claim 17. See details for Claim 17 rejection. Ashe discloses the claimed aspect of a specific window is disposed immediately below a window whose stacking order is a

bottom among windows belonging to a group at a top of the stack of windows, wherein to remove a window from a window group, the RemoveFromGroup method is preferably invoked using the window reference and the window group reference as its arguments. By invoking the RemoveFromGroup method, the address pointer of the removed window is deleted from the fourth entry 824 in the table array 804 associated with the selected window group. The first entry 821 in the table array 804 is then decremented to reflect the current number of windows in the group. Finally, the element 710 corresponding to the removed window is accessed in order to decrement the third entry 733 signifying the current number of window groups to which the window belongs. (Ashe, Page 13, Paragraphs 5-15).

Regarding Claim 34, most of the limitations have been met in the rejection of Claim 17. See details for Claim 17 rejection. Ashe achieves the claimed aspect of Wherein an X window system and a window manager manage a the stack of a the plurality of windows, and the window manager confirms whether or not its a recognized stack conforms to a stack recognized by the window system when receiving a window destruction notification, and, when the recognized stack of the window manager does not conform to the stack recognized by the window system, the window manager performs processing for conforming a the stack recognized by the window system to a the stack recognized by the window manager in RemoveFromGroup (); function, wherein to remove a window from a window group, the RemoveFromGroup method is preferably invoked using the window reference and the window group reference as its

arguments. By invoking the RemoveFromGroup method, the address pointer of the removed window is deleted from the fourth entry 824 in the table array 804 associated with the selected window group. The first entry 821 in the table array 804 is then decremented to reflect the current number of windows in the group. Finally, the element 710 corresponding to the removed window is accessed in order to decrement the third entry 733 signifying the current number of window groups to which the window belongs. (Ashe, Page 13, Paragraphs 5-15).

Furthermore, Ashe discloses a RemoveWindowGroupFromWindowGroup (); function, wherein the RemoveWindowGroupFromWindowGroup method preferably functions to remove a window group from another window group using, as arguments, the references for the base group and the target group. Here, invocation of the method results in the address pointer for the target window being deleted from the fifth entry 825 in the table 804 corresponding to the base group. In addition, the value of the second entry 822 for the base group is decremented signifying that the base group contains one less window group. Finally, the data element 802 corresponding to the target window group is accessed and the value of the third entry 823 is decremented. (Ashe, Page 13, Paragraphs 40-50).

Regarding Claim 35, most of the limitations have been met in the rejection of Claim 17. See details for Claim 17 rejection. Ashe discloses the claimed aspect of an X window system and a window manager manage the stack of a the plurality of windows, and the window manager sets a flag when requesting the window system to

change a the stack, and confirms whether or not its a recognized stack conforms to a stack recognized by the window system only when the flag is set at a reception of a window destruction notification, and, when the recognized stack of the window manager does not conform to the stack recognized by the window system, the window manager performs processing for conforming the stack recognized by the window system to a the stack recognized by the window manager, to thereby to put the flag down in FIG. 7 and FIG. 8, wherein Mark 736 and Mark 827 are illustrated that may be either set or unset during window operations. Applicant should duly note that Mark has a flag function.

Regarding Claim 36, most of the limitations have been met in the rejection of Claim 17. See details for Claim 17 rejection. Ashe discloses the claimed aspect of the collectively arranging includes collecting, when a request for displaying a first window which is not collectively arranged as a group is received from the application program, the first window and a second window group of windows which belong to a same group as the first window and which have been collectively arranged as a group such that a stacking order of the first window is consecutively followed by a stacking order of the second window group in FIG. 8, wherein linked structure of window groups are illustrated.

Regarding Claim 37, Ashe discloses the claimed aspect of a window management program for managing a stack of a plurality of windows which are displayed on a display unit based on one or more application programs the window

management program being recorded on a computer readable medium, in FIG. 3, wherein a Window Manager 305 and Application Program 302 are illustrated.

Ashe discloses the claimed aspect of designating a group of a window from the application program, wherein a window grouping mechanism for creating, organizing and manipulating windows and, more specifically, window groups for display to a user of a computer system is disclosed, by using a series of linked data structures configured to organize information pertaining to all windows defined by an application program executing on the computer system. (See Abstract).

Ashe discloses the claimed aspect of designating a group of the window from the application program in FIG. 7 and FIG. 8, wherein window grouping is illustrated.

Ashe discloses the claimed aspect of receiving a request for displaying the window from the application program, wherein the window manager 205 is a set of software routines within the operating system 204 that is responsible for managing windows 242 displayed on a display screen 235 and viewed by the user during operation of the application program 202. The window manager 205 typically acts in direct response to task commands sent from the application program 202 to the operating system 204 via the API layer 207 as shown by arrow 203. (Ashe, Page 6, Paragraph 30). More specifically, the window manager 205 may further use a graphics system 209 within the operating system to draw the display screen 235. The graphics system 206 stores the information to be displayed directly (via arrow 208) into a screen buffer 210. Under the control of various hardware and software in the computer system 200, the contents of the screen buffer 210 are read out of the buffer 210 and provided,

as indicated schematically by arrow 212, to a display adapter 214. The display adapter 214 contains hardware and software (sometimes in the form of firmware) which converts the information in the screen buffer 210 to a form which can be used to drive the display screen 235 of monitor 232, which is connected to the display adapter 214 by an adapter cable 218. (Ashe, Page 6, Paragraphs 35-45).

Ashe discloses the claimed aspect of collectively arranging the window as a group so as to determine a stacking order of the window in the group when displaying the window in response to the display request, wherein in the collectively arranging step, a stacking order of the window is determined so as to be consecutively followed by a stacking order of a group to which the window belong, with maintaining stacking orders of respective window groups of windows which have been collectively arranged as groups in FIG. 7 and FIG. 8, wherein the Master Application Window List 700 comprises a singly linked list of data elements 710 wherein each data element 710 corresponds to a window 306 (FIG. 3) created by the application program 302. Each data element 710, moreover, is preferably arranged as a table array 720 comprising a series of entries 730 containing fields for storing information related to that window 306. For example, the first entry 731 identifies the class of window being represented, e.g., document, floating or modal. The second entry 732 contains the information regarding the window's appearance as defined by the application program 302. The second entry 732, rather than containing the actual data supporting the window 306, may instead contain a pointer to another location in memory 114 (FIG. 1) where that information is stored. (Ashe, Page 9, Paragraphs 25-40). The third entry 733 in the table array 720 contains a

value indicating the number of window groups to which this window 306 belongs. The significance of this entry 733 is explained in more detail below. The fourth entry 734 signifies whether any window groups are attached to the window 306 and, if so, the identification and memory location of that window group, again, as explained below. The fifth entry 735 in each table array 720 is a forward link to the next element 710 in the list 700 as shown by arrow 738 and the last entry 736 contains a mark that may be either set or unset during window operations. (Ashe, Page 10, Paragraphs 45-50).

Ashe does not teach the claimed aspect of updating the stack order of a window within a window group without altering a stacking order of the groups of windows within the stack of windows. It would be obvious to one of ordinary skill in the art the time of the invention to update the window order within the stack without modifying the stacking order of the other groups, because this would provide modularity and a system composed of modules that each serve a specific purpose in this case display a window within a group and communicate with each other to produce the system's overall behavior.

Regarding Claim 38, Ashe discloses the claimed aspect of window management apparatus in FIG. 2, wherein a Computer 200 is illustrated. Ashe discloses the claimed aspect of for managing, when displaying a plurality of windows on a display unit, a stack of the windows, comprising one or more application programs for displaying one or more windows on the display unit; a window management program for managing a stack of the windows displayed by the one or more application programs and a

processing unit for executing the application programs and the window management program, wherein the application program designates a group of a window with respect to the window management program, and the window management program performs control, when receiving a request for displaying a window from the application program and displaying the window, so as to collectively arrange the window as a group so as to determine a stacking order of the window in the group. This is Claim 17 and 37 related, the rejection of Claims 17 and 37 apply to Claim 38.

Conclusion

The prior art made of record and not relied upon is considered pertinent to applicant's disclosure.

1) Bloomfield, et al., US 5,412,776, 05/02/1995, "Method of generating a hierarchical window list in a graphical user interface".

2) Kamata, et al., US 5,600,346, 02/04/1997, "Multiwindow display control method and apparatus".

3) Orton, et al., US 5,615,326, 03/25/1997, "Object-oriented viewing framework having view grouping".

4) Trueblood, US 5,675,755, 10/07/1997, "Window system preventing overlap of multiple always-visible windows".

5) Trueblood, US 6,031,530, 02/29/2000, "Always-visible window class with overlap prevention".

6) Mairs, et al., US 6,271,839 , 08/07/2001, "Method and system for sharing applications between computer systems".

7) Bloomfield, et al., US 5,412,776, 05/02/1995, "Method of generating a hierarchical window list in a graphical user interface".

8) Minoura, Tadaaki et al., US 20020036661 A1, 03/28/2002, "Method for displaying a window".

9) Robertson, et al., US 6,909,443 , 06/21/2005, "Method and apparatus for providing a three-dimensional task gallery computer interface".

Applicant's amendment necessitated the new ground(s) of rejection presented in this Office action. Accordingly, **THIS ACTION IS MADE FINAL**. See MPEP § 706.07(a). Applicant is reminded of the extension of time policy as set forth in 37 CFR 1.136(a).

A shortened statutory period for reply to this final action is set to expire THREE MONTHS from the mailing date of this action. In the event a first reply is filed within TWO MONTHS of the mailing date of this final action and the advisory action is not mailed until after the end of the THREE-MONTH shortened statutory period, then the shortened statutory period will expire on the date the advisory action is mailed, and any extension fee pursuant to 37 CFR 1.136(a) will be calculated from the mailing date of

the advisory action. In no event, however, will the statutory period for reply expire later than SIX MONTHS from the date of this final action.

Any inquiry concerning this communication or earlier communications from the examiner should be directed to ECE HUR whose telephone number is (571) 270-1972. The examiner can normally be reached on Mon-Thurs 7:30am-5pm.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, WEILUN LO can be reached on 571-272-4847. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

March 7, 2008

Ece Hur
E.H. /e.h.

/Ba Huynh/

Primary Examiner, Art Unit 2179